**“CODE PLAGIARISM”**

**A PROJECT REPORT**

***Submitted by***

**Mr Gour Digvijaysingh and Mr Pathak Swapnil**

***In partial fulfilment for the award of Final Yr. B. Tech Project***

***Of***

**Final Year B. Tech (CSE)**

**IN**

**Computer Science and Engineering**

**At**

****

**SHRI GURU GOBIND SINGHJI INSTITUTE OF ENGINEERING AND TECHNOLOGY, VISHNUPURI, NANDED (MAHARASHTRA STATE) PIN 431606 INDIA.**

**November 2014.**

**CODE PLAGIARISM**

**ABSTRACT**

Plagiarism is one form of academic dishonesty, which is often done by students in programming classes. In a large class, detecting plagiarism manually is both difficult and time-consuming, especially due to the numerous modifications of the source code to conceal the cheating.

Plagiarism detection plays an important role in software security protection and license issues. Source code plagiarism detection method can be classified as string-based, token-based, parse-tree-based and program dependency-based. All of these approaches have certain limitations and can not meet the requirements when the source code is large and may produce false positives. But, parse-tree based detection improves the detection ability and efficiency. This paper describes method based source code detection, which detect the simple plagiarized code like exact match, near exact match and longest common sequence. And also proposes the agent based detection which will perform the detection automatically. Automatic plagiarism detection will be helpful for code clone detection in software industry.

Our aim is to provide a mechanism which will make sure that a student just can simply copy paste any given code just simply from webpages.

Guided by, Project Co-ordinator,

Mr. Abdul Mateen Ms. K.D Joshi

Swapnil Pathak(2011BCS024)

Gour Digvijaysingh(2011BCS047)

**ACKNOWLEDGEMENT**

We would like to express my gratitude and appreciation to all those who gave me the possibility to complete this report. Thanks to our Final Year Project Coordinator, **Ms. K.D Joshi**, whose help, stimulating suggestions and encouragement, helped me to gather all available information on seminar topic.

We would also like to acknowledge with appreciation the role of the staff of **Central Computing Facility**, who gave the permission to use all required software and the necessary resources to complete Plagiarism Testing.

Last but not least, many thanks go to guide of the project, **Mr. Abdul Mateen**, who has given his full effort in guiding me in achieving the goal as well as his encouragement to maintain our progress in track. We would appreciate the guidance given by his that has improved our presentation skills by their comment and tips.

TABLE OF CONTENTS

**CHAPTER TITLE PAGE NO.**

**NO.**

ABSTRACT i

ACKNOWLEDGEMENT iii

1. **Introduction 1**

1.1 What is Code Plagiarism ? 2

1.2 Basic Ways of Plagiarizing code 3

1.3 Why Students Plagiarize ? 4

1.4 How does Plagiarism Detection works ? 5

**2. Specifications 6**

2.1 Software Specifications 6

2.2File Usage **7**

2.3 Third Party Libraries 8

**3. Project Description 9**

3.1 Webpage Extraction module 9

3.2 Program Extraction module 10

3.3 Comparison module 11

**4**. **Conclusion 12**

4.1 Limitations 12

* 1. Final Evaluation

1. **Introduction**
   1. **What is Code Plagiarism?**

In the 1st century, the use of the Latin word plagiarius (literally kidnapper) to denote stealing someone else's work was pioneered by Roman poet [Martial](http://en.wikipedia.org/wiki/Martial), who complained that another poet had "kidnapped his verses." "Plagiary", a derivative of "plagiarus" was introduced into English in 1601 by dramatist [Ben Jonson](http://en.wikipedia.org/wiki/Ben_Jonson) to describe someone guilty of [literary theft](http://en.wikipedia.org/w/index.php?title=Literary_theft&action=edit&redlink=1).

The derived form plagiarism was introduced into English around 1620. The [Latin](http://en.wikipedia.org/wiki/Latin) plagiārius, "kidnapper", and plagium, "kidnapping", has the root plaga ("snare", "net"), based on the [Indo-European root](http://en.wikipedia.org/wiki/Proto-Indo-European_root) \*-plak, "to weave" (seen for instance in [Greek](http://en.wikipedia.org/wiki/Greek_language) plekein, Bulgarian "плета" pleta, Latin plectere, all meaning "to weave").

10 main forms of plagiarism that students commit:

1. Submitting someone’s work as their own.
2. Taking passages from their own previous work without adding citations.
3. Re-writing someone’s work without properly citing sources.
4. Using quotations, but not citing the source.
5. Interweaving various sources together in the work without citing.
6. Citing some, but not all passages that should be cited.
7. Melding together cited and uncited sections of the piece.
8. Providing proper citations, but fails to change the structure and wording of the borrowed ideas enough.
9. Inaccurately citing the source.
10. Relying too heavily on other people’s work. Fails to bring original thought into the text.

Plagiarism is :

* to steal and pass off (the ideas or words of another) as one's own
* to use (another's production) without crediting the source
* to commit literary theft
* to present as new and original an idea or product derived from an existing source
* turning in someone else's work as your own
* copying words or ideas from someone else without giving credit
* failing to put a quotation in quotation marks
* giving incorrect information about the source of a quotation
* changing words but copying the sentence structure of a source without giving credit
* copying so many words or ideas from a source that it makes up the majority of your work, whether you give credit or not (see our section on "fair use" rules)

In other words, plagiarism is an act of fraud. It involves both stealing someone else's work and lying about it afterward.

Plagiarism is intentionally or unintentionally reproducing (copying, rewording, paraphrasing, adapting, etc) work that was produced by another person(s) without proper acknowledgement in an attempt to gain academic benefit. Intentionally or negligently allowing such reproduction to happen may also constitute plagiarism.

Work that can be plagiarized includes: words (language), ideas, findings, writings, graphic representations, computer programs, diagrams, graphs, illustrations, creative work, information, lectures, printed material, electronic material, or any other original work created by someone else.

When it comes to plagiarism, technology has been both a blessing and a curse. Though it has made it easier than ever to find and copy work from others without attribution, it’s also made it easier to track and handle plagiarism when it happens.

With tools that can search billions of documents in seconds and can find matches only a few words in length, it might seem as if plagiarism would be as easily detected as finding information in Google. A matter of merely punching your query and going through the results.

Unfortunately, that isn’t the case.

Plagiarism detectors have a huge limitation and one that isn’t likely to go away any time soon. That limitation is, simply put, that plagiarism detectors can’t actually detect plagiarism and, instead, do something very different altogether.

Objectives of Plagiarism detection are :

* To develop a greater understanding of Author’s rights, plagiarism prevention policies and academic writing practices.
* To explore and develop information resources, technical and other tools and administrative methods used for plagiarism detection and penalising of transgressors.
* To document cases of good practice in plagiarism prevention (including student training, staff consulting, detection tools and administrative methods usage);
* To evaluate effectiveness and impact of policies for prevention of plagiarism;
* To prepare recommendations for improvement of plagiarism prevention policies and tools .

#### 1.2 Basic ways of plagiarizing codes

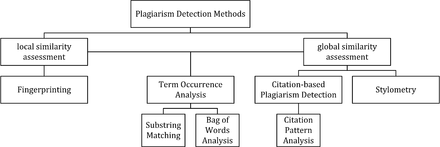
In source-code assignments, students are required to acknowledge the source and authorship of the source-code that was not originally authored by them, within the program source-code (in the format of a comment) and in the appropriate documentation. Here are some examples of plagiarism in source-code assignments identified by Cosma and Joy (2006).

* Reproducing/copying source-code without making alterations and without providing proper acknowledgements.
* Minimally or moderately adapting source-code authored by someone else without providing proper acknowledgements.
* Converting an entire or part of someone else’s source-code to a similar programming language.
* Using code-generating software for creating code without providing acknowledgement of this fact.
* Paying someone else to create the program.
* Collaborating with another student to create a programming assignment when collaboration is not permitted.

Each of the 10 most common types of plagiarism are defined below. The types are ranked in order of severity of intent :

* 1. Clone : Submitting another’s work, word-for-word, as one’s own
  2. CTRL-C :Contains significant portions of text from a single source without alterations.
  3. Find – Replace : Changing key words and phrases but retaining the essential content of the source.
  4. Remix : Paraphrases from multiple sources, made to fit together.
  5. Recycle : Borrows generously from the writer’s previous work without citation
  6. Hybrid : Combines perfectly cited sources with copied passages without citation
  7. Mashup : Mixes copied material from multiple sources
  8. 404 Error : Includes citations to non-existent or inaccurate information about sources.
  9. Aggregator : Includes proper citation to sources but the paper contains almost no original work
  10. Retweet : Includes proper citation, but relies too closely on the text’s original wording and/or structure

Working of plagiarism is as follows :

[](http://en.wikipedia.org/wiki/File:PDS_Classification.png)

##### **Finger printing**

Fingerprinting is currently the most widely applied approach to plagiarism detection. This method forms representative digests of documents by selecting a set of multiple substrings ([n-grams](http://en.wikipedia.org/wiki/N-gram)) from them. The sets represent the fingerprints and their elements are called minutiae. A suspicious document is checked for plagiarism by computing its fingerprint and querying minutiae with a precomputed index of fingerprints for all documents of a reference collection. Minutiae matching with those of other documents indicate shared text segments and suggest potential plagiarism if they exceed a chosen similarity threshold. Computational resources and time are limiting factors to fingerprinting, which is why this method typically only compares a subset of minutiae to speed up the computation and allow for checks in very large collection, such as the Internet.

##### **String matching**

[String matching](http://en.wikipedia.org/wiki/String_matching) is a prevalent approach used in computer science. When applied to the problem of plagiarism detection, documents are compared for verbatim text overlaps. Numerous methods have been proposed to tackle this task, of which some have been adapted to external plagiarism detection. Checking a suspicious document in this setting requires the computation and storage of efficiently comparable representations for all documents in the reference collection to compare them pairwise. Generally, suffix document models, such as [suffix trees](http://en.wikipedia.org/wiki/Suffix_tree) or suffix vectors, have been used for this task. Nonetheless, substring matching remains computationally expensive, which makes it a non-viable solution for checking large collections of documents.

##### **Bag of words**

Bag of words analysis represent the adoption of [vector space retrieval](http://en.wikipedia.org/wiki/Vector_space_model), a traditional IR concept, to the domain of plagiarism detection. Documents are represented as one or multiple vectors, e.g. for different document parts, which are used for pair wise similarity computations. Similarity computation may then rely on the traditional [cosine similarity measure](http://en.wikipedia.org/wiki/Cosine_similarity), or on more sophisticated similarity measures.

##### **Citation analysis**

Citation-based plagiarism detection (CbPD) relies on citation analysis, and is the only approach to plagiarism detection that does not rely on the textual similarity. CbPD examines the citation and reference information in texts to identify similar [patterns](http://en.wikipedia.org/wiki/Pattern) in the citation sequences. As such, this approach is suitable for scientific texts, or other academic documents that contain citations. Citation analysis to detect plagiarism is a relatively young concept. It has not been adopted by commercial software, but a first prototype of a citation-based plagiarism detection system exists. Similar order and proximity of citations in the examined documents are the main criteria used to compute citation pattern similarities. Citation patterns represent subsequences non-exclusively containing citations shared by the documents compared. Factors, including the absolute number or relative fraction of shared citations in the pattern, as well as the probability that citations co-occur in a document are also considered to quantify the patterns’ degree of similarity.\

##### **Stylometry**

[Stylometry](http://en.wikipedia.org/wiki/Stylometry) subsumes statistical methods for quantifying an author’s unique writing styleand is mainly used for authorship attribution or intrinsic CaPD. By constructing and comparing stylometric models for different text segments, passages that are stylistically different from others, hence potentially plagiarized, can be detected.

#### Performance

Comparative evaluations of plagiarism detection systems indicate that their performance depends on the type of plagiarism present (see figure). Except for citation pattern analysis, all detection approaches rely on textual similarity. It is therefore symptomatic that detection accuracy decreases the more plagiarism cases are obfuscated

The design of plagiarism detection software for use with text documents is characterized by a number of factors:

|  |  |
| --- | --- |
| **Factor** | **Description and alternatives** |
| Scope of search | In the public internet, using search engines / Institutional databases / Local, system-specific database. |
| Analysis time | Delay between the time a document is submitted and the time when results are made available. |
| Document capacity / Batch processing | Number of documents the system can process per unit of time. |
| Check intensity | How often and for which types of document fragments (paragraphs, sentences, fixed-length word sequences) does the system query external resources, such as search engines. |
| Comparison algorithm type | The algorithms that define the way the system uses to compare documents against each other. |
| Precision and Recall | Number of documents correctly flagged as plagiarized compared to the total number of flagged documents, and to the total number of documents that were actually plagiarized. High precision means that few [false positives](http://en.wikipedia.org/wiki/False_positives) were found, and high recall means that few [false negatives](http://en.wikipedia.org/wiki/False_negatives) were left undetected. |

Source-code similarity detection algorithms can be classified as based on either

* Strings – look for exact textual matches of segments, for instance five-word runs. Fast, but can be confused by renaming identifiers.
* Tokens – as with strings, but using a [lexer](http://en.wikipedia.org/wiki/Lexical_analysis) to convert the program into [tokens](http://en.wikipedia.org/wiki/Token_(parser)) first. This discards whitespace, comments, and identifier names, making the system more robust to simple text replacements. Most academic plagiarism detection systems work at this level, using different algorithms to measure the similarity between token sequences.
* [Parse Trees](http://en.wikipedia.org/wiki/Parse_tree) – build and compare parse trees. This allows higher-level similarities to be detected. For instance, tree comparison can normalize conditional statements, and detect equivalent constructs as similar to each other.
* [Program Dependency Graphs](http://en.wikipedia.org/wiki/Call_graph) (PDGs) – a PDG captures the actual flow of control in a program, and allows much higher-level equivalences to be located, at a greater expense in complexity and calculation time.
* Metrics – metrics capture 'scores' of code segments according to certain criteria; for instance, "the number of loops and conditionals", or "the number of different variables used". Metrics are simple to calculate and can be compared quickly, but can also lead to false positives: two fragments with the same scores on a set of metrics may do entirely different things.
* Hybrid approaches – for instance, parse trees + [suffix trees](http://en.wikipedia.org/wiki/Suffix_tree) can combine the detection capability of parse trees with the speed afforded by suffix trees, a type of string-matching data structure

**1.3 Why do Students Plagiarise?**

Students plagiarise for a variety of reasons and it is important to consider these before reviewing detection and prevention so they can be addressed. It is also worth remembering that a combination of reasons may affect a student’s decision to plagiarise. In this instance, no distinction has been made between the plagiarism of external sources and plagiarism of their peers’ work (often referred to as collusion). The following are ten examples of reasons why students might plagiarise.

1) **Bad time management skills**

Perhaps the most common reason people plagiarise is bad time management skills. Having left it to the last minute to complete an assignment they panic and try to find the quickest solution. External pressures affecting much more of the student population, such as the need to work or care for children, may make the situation worse.

**2) Unable to cope with the work load**

This is similar to bad time management, but this problem lies with the student’s timetable and assignments from multiple modules clashing.

**3) The tutor doesn’t care, why should I?**

If the student senses that the academic is not interested in the subject or the student’s learning then the student is less inclined to care. This apathy by the lecturer can be shown in a multitude of ways such as showing no enthusiasm for the subject, providing handouts that have obviously been used for years or assignments that seem dated.

**4) External pressure to succeed**

In the US, statistics have shown that one of the main reasons people resort to plagiarism is the need to keep up a grade average. Although this does not appear to be an issue in the UK, there may be external pressures such as parental and cultural expectations that make students feel they have to plagiarize to achieve.

**5) Lack of understanding**

The most common cause of minor plagiarism is a lack of understanding of how to cite material from other sources.

**6) I can’t do this!**

If a student is faced with an assignment they feel is completely beyond their ability they may feel they have no option but to copy the answers. However, this may have more to do with a lack of clarity in the assignment specifications than a student’s ability. This problem is often linked to bad management skills: it is human nature to leave until last the things we either consider hard or unpleasant.

**7) I want to see if I can get away with it**

Students may be motivated to see if they can get away with plagiarism for several reasons. If they are trying to test the institution and/or academic, it is likely that, whatever prevention methods are put into place, this small proportion of students will always attempt to plagiarise. In fact there is an agreement that the more visible prevention methods are the more challenging for students. However, particular problems arise if the institutional policy encourages students to plagiarise merely to see if they can get away with it. If a student has left an assignment until the last minute, knowing that the penalty for plagiarism will simply be to resubmit the work,, they are in a win-win situation. Either they won’t get caught or they will effectively be given an extension.

**8) I don’t need to learn this, I only need to pass it**

If a student is not motivated to take part in the educational process or does not appreciate that they need to acquire the knowledge to continue their education, they may be inclined to take the quickest route to success.

**9) But you said work together!**

Most people in the project identified collusion as far bigger problem than plagiarism from printed material or the web. As noted in the introduction to this section, no distinction has been made between the plagiarism of external sources and plagiarism of peers’ work. In this instance the term collusion has been used to describe a situation when students have been asked to work together on an assignment and have presented the same text. Obviously in some cases the assignment specification allowed for this; if not the work will be regarded as plagiarism. It is important that the specification makes clear what is expected so students are aware if individual or joint assignments are required.

**10) But that would insult the experts in the field**

Finally, there is the issue of cultural differences in learning and presentation styles. In some countries it is customary to include material from experts in the field without citation. Although all students must work under their institution’s regulations it is worth taking this into account when training students in study skills.

1. **Specifications**
   1. **Software Specifications**

**Front End** : Java Swing

**Back End :** Java , File System

**OS Compatible :** All Java Enabled OS are compatible

**Hardware Requirements :** All Hardware systems that can support Java

* 1. **File System Usage**

This project makes use of file system by storing some files in it. There are 3 files involved for storing :

1. Link.txt : Link.txt file is used to store limited number of links that are involved or related with program topic of target file. These links contain programs that are probably used for plagiarism in target program. These links are nothing but websites that are accessible by program which are later used for extracting complete webpage into a file.
2. Sample.txt : For each accessible link , a sample.txt file is generated which contains the complete document form of website specified in the link.Sample.txt is later sampled to extract programs out of it. It is completely tokenized so as to provide ease in generating programs out of it.
3. Program.txt : For each sample.txt , a program.txt is generated which is nothing but programs synthesized out of webpage content in sample.txt .

**2.3 3rd Party Liabraries:**

# 2.3.1 jsoup: Java HTML Parser

jsoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods.

jsoup implements the [WHATWG HTML5](http://whatwg.org/html) specification, and parses HTML to the same DOM as modern browsers do :

* 1. scrape and [parse](http://jsoup.org/cookbook/input/parse-document-from-string) HTML from a URL, file, or string
  2. find and extract data, using DOM traversal or CSS selectors
  3. [manipulate](http://jsoup.org/cookbook/modifying-data/set-html) the HTML elements, attributes, and text
  4. [clean](http://jsoup.org/cookbook/cleaning-html/whitelist-sanitizer) user-submitted content against a safe white-list, to prevent XSS attacks
  5. [output](http://jsoup.org/apidocs/org/jsoup/select/Elements.html#html()) tidy HTML

jsoup is designed to deal with all varieties of HTML found in the wild; from pristine and validating, to invalid tag-soup; jsoup will create a sensible parse tree.

## Example

Fetch the Wikipedia homepage, parse it to a DOM, and select the headlines from the In the news section into a list of Elements (online sample):

Document doc = Jsoup.connect("http://en.wikipedia.org/").get();

Elements newsHeadlines = doc.select("#mp-itn b a");

## Open source

jsoup is an open source project distributed under the liberal [MIT license](http://jsoup.org/license). The source code is available at [GitHub](http://github.com/jhy/jsoup/)

**2.3.2 JSON:**

**JSON** (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the [JavaScript Programming Language](http://javascript.crockford.com/), [Standard ECMA-262 3rd Edition - December 1999](http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf). JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON is built on two structures:

* A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
* An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

In JSON, they take on these forms:

An *object* is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma).



An array is an ordered collection of values. An array begins with [ (left bracket) and ends with ] (right bracket). Values are separated by , (comma).



A value can be a string in double quotes, or a number, or true or false or null, or an object or an array. These structures can be nested.



A string is a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes. A character is represented as a single character string. A string is very much like a C or Java string.



A number is very much like a C or Java number, except that the octal and hexadecimal formats are not used.



Whitespace can be inserted between any pair of tokens. Excepting a few encoding details, that completely describes the language.

**2.3.3 Gson:**

Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of.

There are a few open-source projects that can convert Java objects to JSON. However, most of them require that you place Java annotations in your classes; something that you can not do if you do not have access to the source-code. Most also do not fully support the use of Java Generics. Gson considers both of these as very important design goals.

**Gson Goals**

* Provide simple toJson() and fromJson() methods to convert Java objects to JSON and vice-versa
* Allow pre-existing unmodifiable objects to be converted to and from JSON
* Extensive support of Java Generics
* Allow custom representations for objects
* Support arbitrarily complex objects (with deep inheritance hierarchies and extensive use of generic types)

**Features:**

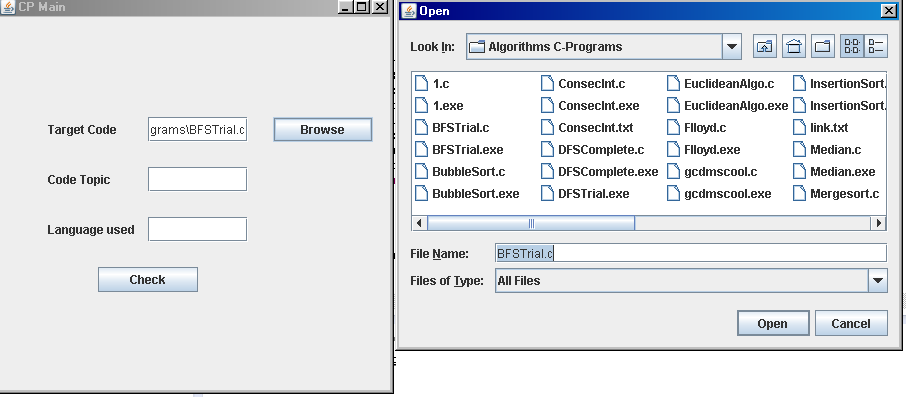
* Gson can handle collections, generic types and nested classes (including inner classes, this can not be done by default though)
* When deserializing, Gson is navigating the type tree of the object, which is being deserialized. This results in ignoring extra fields present in the JSON input.
* User can write a custom serializer and/or deserializer so that they can control the whole process and even (de)serialize instances of classes for which the source code is not accessible.
* User can write an InstanceCreator which allows them to deserialize instances of classes without a defined no-args constructor.
* Gson is highly customizable, you can specify:
* Compact/pretty printing (whether you want compact or readable output)
* How to handle null object fields - by default they are not present in the output
* Rules of what fields are intended to be excluded from (de)serialization
* How to convert Java field names

**3 Project Description**

* 1. **Webpage Extraction Module :**

Webpage extraction deals with extraction of webpage contents from given set of links. Process undergoes down as follows :

Step 1 : First user provides path of target file to be checked for plagiarism by using GUI and FileChooser Menu. Path is extracted from checkbox number 1 in following.



Code topic and Language used are also taken as input by user. Using Code topic and Language, we generate URL and perform a Google Search at back end. Then we extract links by using code and write it line by line into file called link.txt.

Code involved is as follows :

import javax.swing.\*;

import java.awt.event.\*;

import java.io.\*;

import java.net.\*;

import org.jsoup.\*;

import org.jsoup.nodes.\*;

import com.google.gson.Gson;

import java.util.\*;

public class TestingMight implements ActionListener {

JTextField jt1,jt2,jt3;

JFrame main,f;

JFileChooser jfc;

TestingMight()

{

main=new JFrame("CP Main");

main.setLayout(null);

jfc = new JFileChooser();

JLabel jl3 = new JLabel("Target Code");

JLabel jl1 = new JLabel("Code Topic");

JLabel jl2 = new JLabel("Language used");

JButton jb = new JButton("Check");

JButton jb2 = new JButton("Browse");

jt1 = new JTextField();

jt2 = new JTextField();

jt3 = new JTextField();

jl1.setSize(100,25);

jl2.setSize(120,25);

jl3.setSize(100,25);

jt1.setSize(100,25);

jt2.setSize(100,25);

jt3.setSize(100,25);

jb.setSize(100,25);

jb2.setSize(100,25);

jl3.setLocation(50,100);

jl1.setLocation(50,150);

jl2.setLocation(50,200);

jt3.setLocation(150,100);

jt1.setLocation(150,150);

jt2.setLocation(150,200);

jb.setLocation(100,250);

jb2.setLocation(275,100);

jt1.addActionListener(this);

jt2.addActionListener(this);

jt3.addActionListener(this);

jb.addActionListener(this);

jb2.addActionListener(this);

main.add(jl1);

main.add(jl2);

main.add(jl3);

main.add(jt1);

main.add(jt2);

main.add(jt3);

main.add(jb);

main.add(jb2);

main.setSize(400,400);

main.setVisible(true);

}

public void actionPerformed(ActionEvent e) {

int c=0;

File target= null;

FileOutputStream fos;

BufferedWriter bw;

try

{

String str = e.getActionCommand();

if(str.equals("Check"))

{

main.setVisible(false);

f = new JFrame("Result");

f.setLayout(null);

f.setSize(600,600);

String s1=jt1.getText();

String s2=jt2.getText();

String s3=jt3.getText();

String tmp;

String google = "http://ajax.googleapis.com/ajax/services/search/web?v=1.0&q=";

String search = s1+" using "+s2+" language";

String charset = "UTF-8";

URL url = new URL(google + URLEncoder.*encode*(search, charset));

Reader reader = new InputStreamReader(url.openStream(), charset);

GoogleResults results = new Gson().fromJson(reader, GoogleResults.class);

target = new File(s3);

File links = new File(target.getParent()+"/link.txt");

fos = new FileOutputStream(links);

bw = new BufferedWriter(new OutputStreamWriter(fos));

for(int i=0;i<results.getResponseData().getResults().size();i++)

{

bw.write(results.getResponseData().getResults().get(i).getUrl());

bw.newLine();

}

}

import java.util.List;

public class GoogleResults {

private ResponseData responseData;

public ResponseData getResponseData() { return responseData; }

public void setResponseData(ResponseData responseData) { this.responseData = responseData; }

public String toString() { return "ResponseData[" + responseData + "]"; }

static class ResponseData {

private List<Result> results;

public List<Result> getResults() { return results; }

public void setResults(List<Result> results) { this.results = results; }

public String toString() { return "Results[" + results + "]"; }

}

static class Result {

private String url;

private String title;

public String getUrl() { return url; }

public String getTitle() { return title; }

public void setUrl(String url) { this.url = url; }

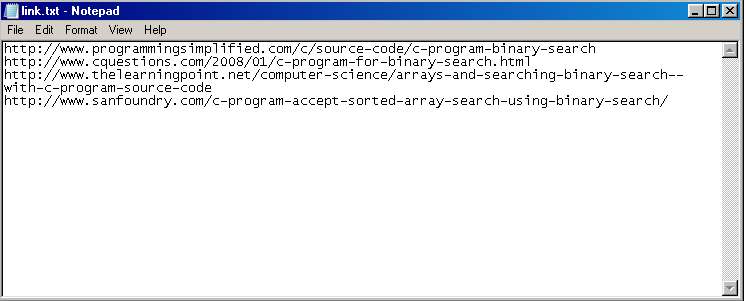
public void setTitle(String title) { this.title = title; }

public String toString() { return "Result[url:" + url +",title:" + title + "]"; }

}

}

Sample link.txt :



Step 2 : After extracting links as above we perform reading from individual links. For we make use of :

* 1. URL Class to read basic HTML content of link.
  2. JSoup Parser to convert HTML into Web document format.

Code involved is :

FileInputStream fis = new FileInputStream(links);

BufferedReader br = new BufferedReader(new InputStreamReader(fis));

c=0;

while((tmp=br.readLine())!=null)

{

URL site = new URL(tmp);

Document doc = null;

try

{

doc = Jsoup.parse(site,3\*1000);

}

catch(HttpStatusException hse)

{

continue;

}

catch(SocketTimeoutException hse)

{

continue;

}

c++;

String info=doc.body().text();

File sample = new File(target.getParent()+"/sample"+c+".txt");

fos = new FileOutputStream(sample);

bw = new BufferedWriter(new OutputStreamWriter(fos));

StringTokenizer st = new StringTokenizer(info);

while (st.hasMoreTokens())

{

bw.write(st.nextToken());

bw.newLine();

}

bw.close();

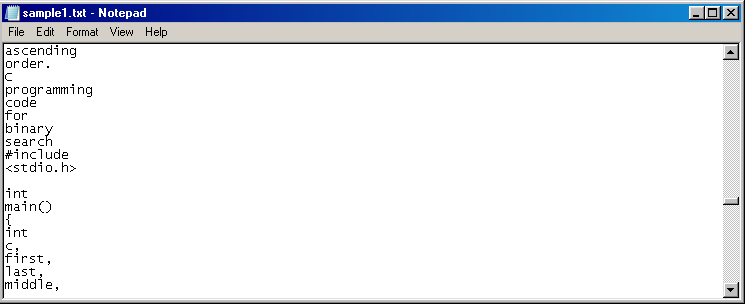
}

fos.close();

fis.close();

}

The result of this code is sample.txt which contains the contents of webpage contents involved



* 1. **Program Extraction Module**

Once Webpage contents are extracted, we need to sample code out of that webpage contents. For that we developed a module which follows a brute force approach.

Its workflow is as follows :

Step 1 : First read sample.txt file

Step 2 : For corresponding sample.txt file generate a program.txt file which contains set(s) of sampled program from the webpage content of given sample file.

Code involved is as follows :

import java.io.\*;

import java.util.\*;

public class CProgramizer {

public static void main(String[] args) throws Exception{

int cp=0,cb=0;

boolean f =false;

BufferedReader br = new BufferedReader(new InputStreamReader(System.*in*));

BufferedReader br2 = new BufferedReader(new InputStreamReader(new FileInputStream(new File(br.readLine()))));

String st;

while((st=br2.readLine())!=null)

{

if(st.equals("Â "))

continue;

if(st.equals("#include")||st.contains("#include"))

f=true;

if(f)

{

int ck=0;

while((ck=st.indexOf('{',ck))!=-1)

{

cb++;

ck++;

}

ck=0;

while((ck=st.indexOf('(',ck))!=-1)

{

cp++;

ck++;

}

ck=0;

while((ck=st.indexOf(')',ck))!=-1)

{

cp--;

ck++;

}

ck=0;

while((ck=st.indexOf('}',ck))!=-1)

{

cb--;

ck++;

}

if(st.equals("<stdio.h>")||st.equals("{")||st.equals("}")||st.equals("\*/")||st.contains("#include<")||st.equals("else"))

System.*out*.println(st);

else

{ if((st.indexOf('{')==st.length()-1)&&(st.length()!=1))

{

System.*out*.println(st.substring(0, st.indexOf('{')));

System.*out*.println("{");

}

else

{

if((st.charAt(st.length()-1)==';')&&(cp==0))

System.*out*.println(st);

else

{

if((st.charAt(st.length()-1)==')')&&(cp==0))

System.*out*.println(st);

else

System.*out*.print(st+" ");

}

}

}

}

if(st.equals("}")&&(cb==0))

{

st=br2.readLine();

if(st.equals("int")||st.equals("void")||st.equals("float")||st.equals("char")||st.equals("long")||st.equals("short")||st.equals("double")||st.equals("struct"))

{

System.*out*.print(st+" ");

continue;

}

else

{

st=br2.readLine();

if(st.equals("int")||st.equals("void")||st.equals("float")||st.equals("char")||st.equals("long")||st.equals("short")||st.equals("double")||st.equals("struct"))

{

System.*out*.print(st+" ");

continue;

}

else

f=false;

}

}

}

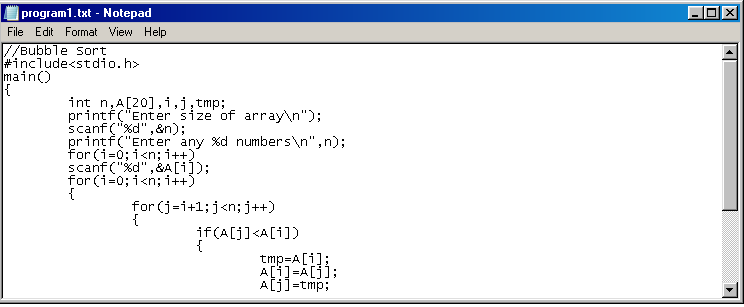
br2.close();

}

}

So after executing this code a program.txt file is generated which contains programs is an appropriate syntactical form.

Sample program.txt file looks like :



* 1. **Comparison Module :**

After we extract all program.txt files , we start comparing each individual programs in program.txt with the target code.

There are 2 types of comparisons involved :

1. Word to word comparison : In word to word comparison , we line by line read line program and directly compare both programs if they are exactly same or not.

For this we take 2 arguments :

* 1. 1st argument = Target file
  2. 2nd argument = program.txt file

We read a line from 1st argument and check it from 2nd argument, if both of them got tallied then we read next line from 1st argument. Irrespective of tallying, 2nd argument always reads a line out of its file.

If 1st argument is out of lines to read then we easily can detect or say that code is copied word to word from site, since otherwise it wouldn’t proceed further.

1. Algorithm Comparison : Sometimes students copy the basic structure of the program but change the name of variables to avoid easy detection of copying. So to detect this, we make use of Algorithm Comparison.

Algorithm Comparison works on principle that, irrespective of variable name changes, the position of declarations, conditional statements, looping statements and use of preprocessor directives are always same.

Legends used are :

P : Preprocessor directives

S : Statements

C : Conditional Statements

L : Looping Statements

Then we encode the two programs using above legends and then we compare encoded string to check for plagiarism. If the encoded string matches, then algorithm is definitely copied.

Code involved is :

import java.io.\*;

public class Comparator {

public static void main(String[] args) throws Exception

{

BufferedReader br = new BufferedReader(new InputStreamReader(System.*in*));

System.*out*.println("Give complete path of file1");

String p1=br.readLine();

System.*out*.println("Give complete path of file2");

String p2=br.readLine();

File f1 = new File(p1);

File f2 = new File(p2);

FileInputStream fis1 = new FileInputStream(f1);

FileInputStream fis2 = new FileInputStream(f2);

BufferedReader bf1 = new BufferedReader(new InputStreamReader(fis1));

BufferedReader bf2 = new BufferedReader(new InputStreamReader(fis2));

String s1=bf1.readLine(),s2;

while((s2=bf2.readLine())!=null)

{

try

{

System.*out*.println(s1);

System.*out*.println(s2);

if(s2.contains(s1)||s2.equals(s1)||s1.contains(s2))

s1=bf1.readLine();

}

catch(NullPointerException npe)

{

System.*out*.println("Code is Word to Word Copied");

break;

}

}

if(bf1.readLine()==null)

System.*out*.println("Code is Word to Word Copied");

else

System.*out*.println("Code is not Word to Word Copied");

bf1.close();

bf2.close();

fis1.close();

fis2.close();

fis1 = new FileInputStream(f1);

fis2 = new FileInputStream(f2);

bf1=new BufferedReader(new InputStreamReader(fis1));

bf2=new BufferedReader(new InputStreamReader(fis2));

String code1="",code2="";

code1=*Encoder*(bf1);

code2=*Encoder*(bf2);

System.*out*.println(code1);

System.*out*.println(code2);

if(code2.contains(code1))

System.*out*.println("Algorithm is copied");

}

static String Encoder(BufferedReader br) throws IOException

{

String c="",s;

while((s=br.readLine())!=null)

{

if(s.equals(" "))

continue;

if(s.contains("#"))

c=c.concat("P");

else

{

if(s.contains("int ")||s.contains("float ")||s.contains("char ")||s.contains("void ")||s.contains("long ")||s.contains("double ")||s.contains(" int ")||s.contains(" float ")||s.contains(" char ")||s.contains(" void ")||s.contains(" long ")||s.contains(" double "))

c=c.concat("D");

else

{

if(s.contains("if")||s.contains("else"))

c=c.concat("C");

else

{

if(s.contains("for")||s.contains("while"))

c=c.concat("L");

else

c=c.concat("S");

}

}

}

}

return c;

}

}

Workflow of project

**4 Conclusion**

**4.1 Limitations**

Since plagiarism detection tools can only detect copying, or more specifically similar phrases, there are two areas where they are particularly weak.

**Non-Verbatim Plagiarism:** Plagiarism that involves the rewriting, translating or otherwise redrafting the text can’t be detected. This can be difficult to get away with as most plagiarism detectors are extremely sensitive, but since plagiarism detectors don’t analyze the content of the work, just the words, it can’t see if you lifted the idea or information if you didn’t also lift the words. This is a common problem in academia, which treats this kind of plagiarism equally as seriously as verbatim plagiarism.

**Common Phrasing/Attributed Use:** Second, though many plagiarism checkers will make an attempt to separate out attributed use, given the variety of attribution styles it isn’t always possible. Also, given how common some phrases are in the English language, many plagiarism checkers will report matches that are actually just coincidence.

In short, plagiarism detection tools are just machines and they can make mistakes. However, that is true with any tool as, for example, you don’t discard Microsoft Word because you can make a typo.

Also, like any other tools, plagiarism checkers are useless without humans to use them intelligently, which is the biggest problem such tools have.

The answer to all of this is simple, the decision as to what is and what is not plagiarism should be left to human beings. Humans are the only ones who can detect non-verbatim plagiarism and are the only one who can make determinations about the likelihood that the matches are coincidence and the whether the attribution was adequate or not.

Professors who have a hard rule about papers not being more than X% matching or authors who don’t let others copy more than X number of words before seeking legal action aren’t fighting plagiarism, but are doing more to confuse the issue.

While bright line rules are always tempting because they are easy to remember and follow, with plagiarism, there are few such rules and you can’t turn your judgment over to a machine.

**4.2 Limitation of this Project :**

There are few serious drawbacks involved in this project :

* 1. In GoogleSearch results of project , not more than 4 links are extracted through modules.
  2. Project is based only for 1 language i.e C Language.
  3. There is high probability of encountering :
     1. HTTPStatusException : It occurs sometimes when website don’t allow their content to be illegally accesed(Status 403).
     2. SocketTimeoutException : This Exception is fired when there is weakness in signal of network connectivity.
     3. NullPointerException : Sometimes in websites while copying and comparing programs, null string appears almost randomly.
  4. Sometimes in word to word comparison , string comparison fails in java, because when we copy directly from website, some stray elements strings which are undisplayable are also introduced thus failing the comparison.
  5. CProgramizing code is way too brute force.

**References :**

1. <https://www.plagiarismtoday.com/dmca-contact-information/>
2. <https://www.plagiarism.org/>
3. <http://en.wikipedia.org/wiki/Plagiarism/>
4. <https://www.ecampusnews.com/>